OHJELMOINTIRAJAPINNAN TOTEUTUS PELOTON.FI -VERKKOPALVELUUN

Sauli Rusanen

Opinnäytetyö
Joulukuu 2010
Tietojenkäsittelyn koulutusohjelma
Proakatemian suuntautumisvaihtoehto
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ
Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Proakatemian koulutusohjelma
RUSANEN, SAULI: Ohjelmointirajapinnan toteutus Peloton.fi -verkkopalveluun
Opinnäytetyö 38 s., liitteet 4 s.
Joulukuu 2010

Opinnäytetyö käsittelee ohjelmointirajapinnan toteuttamista verkkopalveluun. Toimeksiantajana oli Peloton Liikuntakerho Oy. Tavoitteena oli Peloton Liikuntakerhon verkkopalvelun laajentaminen ohjelmointirajapinnan avulla.

Aluksi työssä tarkastellaan ohjelmistoarkkitehtuureja sekä ohjelmointirajapintojen sijoittumista arkkitehtuureihin. Lisäksi tarkastellaan WWW-sovelluspalveluita sekä erilaisia tapoja toteuttaa näitä.

Toteutusosiossa tarkastellaan kuinka toimeksiantona ollut rajapinta käytännössä toteutettiin. Tarkasteltavia asioita ovat toteutustekniikat, dokumentointi, tunnistautuminen sekä ohjelmointirajapinnan käytännön arkkitehtuuri.

Lopuksi opinnäytetyössä pohditaan toteutuksen onnistumista ja mahdollista jatkokehitystä. Työssä pohditaan myös kuinka hyödyllinen toteutettu ohjelmointirajapinta on ja mitä mahdollisuuksia se avaa verkkopalvelulle.

Asiasanat: Ohjelmointirajapinta, verkkopalvelu, WWW-sovelluspalvelu.

ABSTRACT
Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree of Business Information Systems
Specialization Option of Proacademy
RUSANEN, SAULI: Implementing an Application Programming Interface for Peloton.fi-online Service
Bachelor's thesis 38 pages, appendices 4 pages
December 2010

This thesis covers implementing an API (Application Programming Interface) to an online service. The client was Peloton Liikuntakerho Oy. The goal was to extend the online services by adding an API.

First, this thesis looks into software architectures and how APIs take place in software architectures. It also looks into Web Services and different ways of implementing them.

In the chapter 3, this thesis covers the practical implementation of an API. Matters are reviewed are technologies, documentation, authorization and practical architectures of an API.

Finally, the thesis discusses how successful the implementation was and how the API could be developed further. The thesis also discusses the benefits of the implemented API and what possibilities it gives for an online service.

Key words: Application Programming Interface, Web Services, API.

SISÄLLYS

TERMIT JA LYHENTEET

Termien selitykset on haettu TEPA-termipankista, joka on Sanastokeskus TSK:n ylläpitämä ja kokoama termipankki.

Termi / Lyhenne	Selitys
API	API (application program interface) eli ohjelmointirajapinta muodostaa kahden ohjelman, esimerkiksi käyttöjärjestelmän ja sovellusohjelman välisen rajapinnan, jonka tarjoamat valmiit palvelut helpottavat ohjelmoijan työtä.
Atom	Internet Engineering Task Force (IETF) -järjestön alaisuudessa toimivan Atompub Working Groupin kehittämä verkkosyöte
Web tai WWW	HTTP-yhteyskäytäntöön perustuva Internetin palvelujärjestelmä, jonka avulla käyttäjät voivat lukea Internetiin sijoitettuja tiedostoja. Tässä työssä käytetään yleensä termiä Web. Termiä WWW käytetään virallisissa käännössuosituksissa.
Olio	Tietojärjestelmässä yksilöitävissä oleva kokonaisuus, jolla on tila ja käyttäytyminen
XML	Extensible Markup Language, SGML-kielestä erityisesti internetkäyttöä varten rajattu merkintäkieli, joka on helposti laajennettavissa.

1 JOHDANTO

Opinnäytetyöni käsittelee API:n toteuttamista toimeksiantajana toimineen Peloton Liikuntakerho Oy:n verkkopalveluun. Toimeksiannon idea syntyi keskustellessa toimeksiantajan kanssa. Alkuperäisenä ajatuksena oli kehittää palvelua toteuttamalla mobiiliasiakasohjelma palvelun käyttäjille. Mobiililaitteita on kuitenkin olemassa niin monella eri alustalla, että asiakasohjelman toteuttaminen tietylle mobiilialustalle ei olisi välttämättä tuonut lisäarvoa kovinkaan monelle käyttäjälle.

Tavoitteena oli tuoda palvelu saataville entistä paremmin yhä useammalle päätelaitteelle ja alustalle. Toimeksiantajan kanssa asiasta keskustellessamme saimme ajatuksen ohjelmointirajapinnasta, jota voisimme hyödyntää palvelun laajentamiseen useille päätelaitteille. Koska avointen ohjelmointirajapintojen tarjoaminen verkkopalveluista mielletiin myös nousevaksi trendiksi (esimerkiksi Facebook, Google, Twitter, Flickr, del.icio.us jne.), otettiin opinnäytetyöni aiheeksi avoimen ohjelmointirajapinnan toteuttaminen Pelottoman verkkopalveluun.

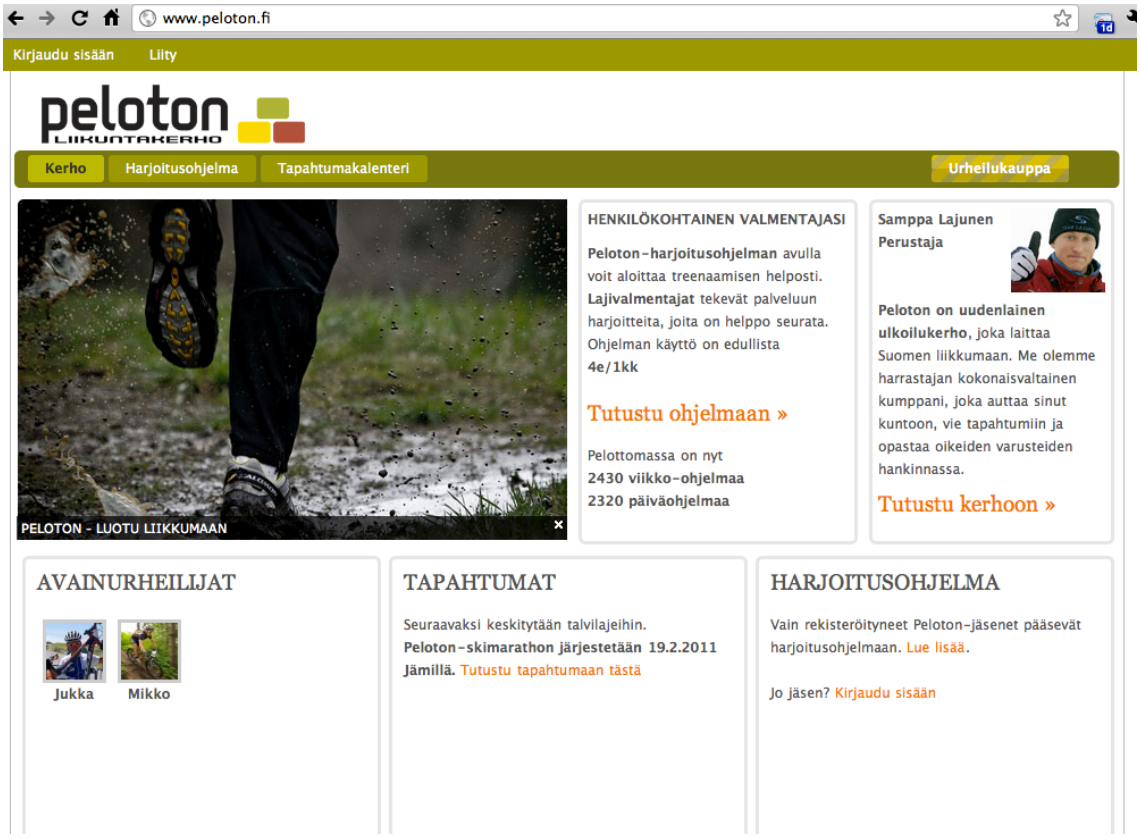
Opinnäytetyön tarkoituksena on toteuttaa yksinkertainen rajapinta, jonka avulla voidaan toteuttaa hakuja verkkopalvelun tietokantaan. Opinnäytetyössäni tarkastellaan ohjelmistoarkkitehtuurin näkemys rajapintoihin, sekä erilaisia tekniikoita toteuttaa ohjelmointirajapintoja verkossa.

Opinnäytetyössäni käydään läpi ohjelmointirajapinnan toteuttaminen ja siihen liittyvät ongelmat ja ratkaisut. Tarkoituksena on avata ohjelmointirajapinnan toteutusprosessia toteuttajan näkökulmasta. Toteutuksesta avataan ajatuksia ratkaisuiden taustalla sekä havainnollistetaan kuvin ja kaavioin, kuinka ohjelmointirajapinta käytännössä toimii.

Opinnäytetyöni on kirjoitettu olettaen, että lukija ymmärtää verkko-ohjelmoinnin peruskäsitteet ja ymmärtää tietojenkäsittelyn termistöä.

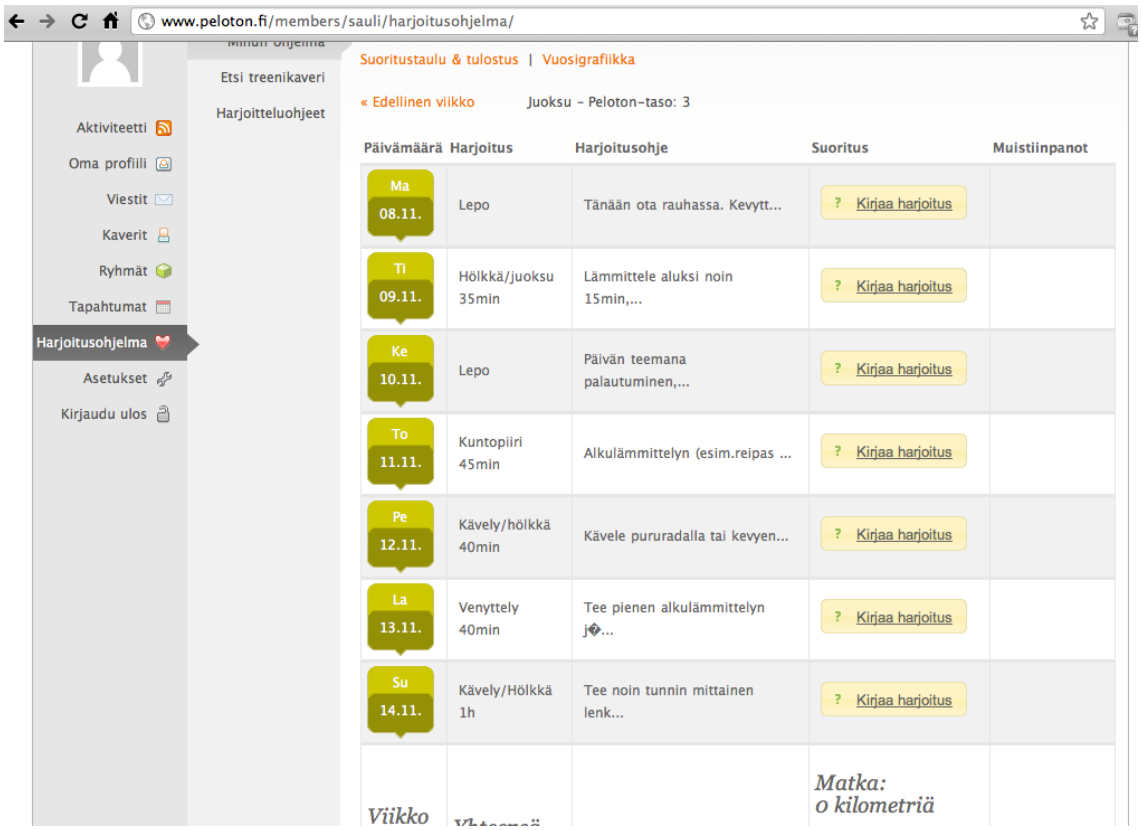
1.1 Toimeksiantajan esittely

Toimeksiantajana toimineen yrityksen nimi on Peloton Liikuntakerho Oy. Yritys omistaa ja ylläpitää verkkopalvelua nimeltä Peloton Liikuntakerho, josta käytän tästä eteenpäin nimitystä Peloton. Peloton on Webbissä toimiva liikuntakerho, jonka ajatuksena on olla harrastajan kokonaisvaltainen kumppani. Palvelu sisältää kolme erillistä osiota: harjoitusohjelma auttaa kuntoilussa, tapahtumakalenterista löytyvät liikuntatapahtumat ja verkkokaupasta löytyvät tarvittavat välineet harrastamiseen. Palvelun etusivu on kuvassa 1.



KUVA 1: Peloton verkkopalvelun etusivu.

Harjoitusohjelma on maksullinen palvelu, johon käyttäjä kirjautuu sisään ja luo oman profiilinsa. Profiilia varten määritetään palveluun oma kuntotaso, joita löytyy välillä rapakuntoinen (1) ja kansainvälisen tason huippu-urheilija (8). Lisäksi palveluun määritetään oma liikuntalaji yhdestätoista eri vaihtoehdosta. Kriteereiden valinnan jälkeen käyttäjä saa haltuunsa Peloton Liikuntakerhon ammattiurheilijoiden luoman harjoitusohjelman, joka tuotetaan kuntotason ja valitun liikuntalajin mukaan. Kuvassa 2 on kuvakaappaus harjoitusohjelma-sivulta.

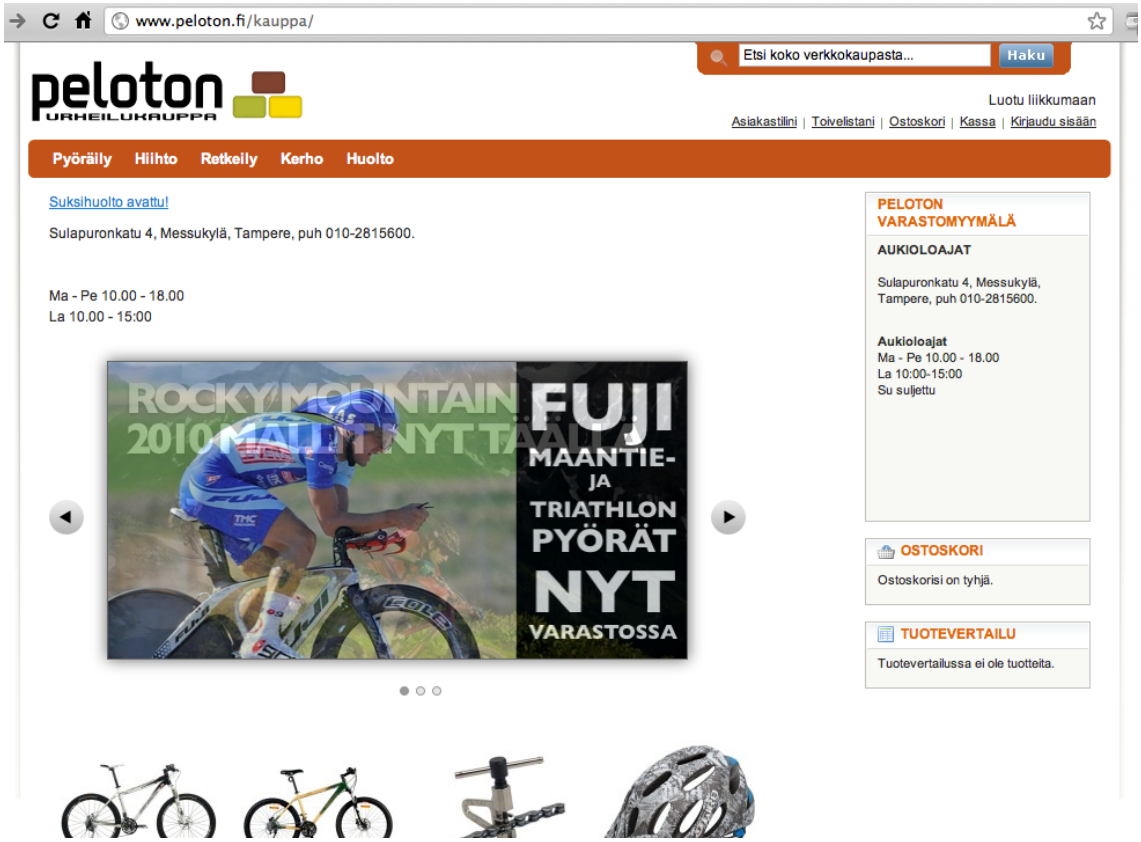


KUVA 2: Harjoitusohjelma osio Peloton-verkkopalvelusta.

Tapahtumakalenterissa ilmoitetaan liikuntatapahtumista, joissa Peloton Liikuntakerho on mukana (kuva 3). Verkkopalveluun kirjautuneet käyttäjät voivat ottaa osaa tapahtumiin ja löytää lisää tietoa tapahtumasta tapahtumakalenterin kautta. Tapahtumat täydentävät harjoitusohjelma-osiota luomalla välietappeja harjoitteluun.

KUVA 3: Tapahtumakalenteri

Peloton-verkkopalveluun sisältyy myös urheiluvälineisiin keskittynyt verkkokauppa (Kuva 4). Kauppa täydentää verkkopalvelua tarjoamalla myyntiin urheiluun ja harjoitteluun tarvittavia laitteita, kuten esimerkiksi pyöräily-, retkeily- ja hiihtotarvikkeita.



KUVA 4: Kuvakaappaus verkkokauppa osiosta

2 TAUSTA

2.1 Ohjelmistokehityksen historiaa

Toiminnallisten sovellusten ymmärtäminen ja toteuttaminen on kehittynyt aikojen saatossa yksinkertaisien ja alkukantaisten välineiden kautta kohti laajempia kokonaisuuksia. Nykyisin voidaan puhua valmiita perusarkkitehtuureja toteuttavista ohjelmistoalustoista. Aluksi kokonaiset sovellukset nähtiin lähinnä rekistereihin ja muistipaikkoihin tallennettujen lukujen ja merkkijonojen käsittelynä. Tämän tyylisen yksinkertaisen rekisteri- ja muistipaikka-ajattelun jälkeen kieliin alettiin kehittää rakenteisia tietotyyppejä, joita aiemmasta toiminnasta alettiin tunnistaa. Samaan tyyliin tunnistettiin toimenpiteitä, joita näille rakenteille tuli toteuttaa; näitä esitettiin kielissä aliohjelmina. Kun sovelluksista alettiin nähdä suurempia käsitteitä, joihin liittyi sekä tietoa että käyttäytymistä, alettiin kieliin kehittää tietoabstraktioita eri muodoissa. (Koskimies & Mikkonen, 2005, 15–16.)

Tietoabstraktion kehittämisen jälkeen havaittiin tarve suurempien kokonaisuuksien muodostamiselle loogisesti yhteenkuuluvista palveluista. Näitä esitettiin tietyt rajapinnat toteuttavina komponentteina. Myöhemmin alkoi yksittäisen sovelluksen käsite jäädä epäoleelliseksi toteutuksen kannalta, kun eri sovellukset alkoivat käyttää samoja komponentteja ja kirjastoja. Kun ymmärrettiin, että monilla erityyppisilläkin ohjelmistoilla saattoi olla samanlainen perusarkkitehtuuri, alettiin kehittää ohjelmistoalustoja. Ymmärrettiin myös, että eri sovellukset saattoivat olla keskenään samanlaisia joko toteutusrakenteen tai toiminnallisuuden kannalta. Näin alettiin tunnistaa sovellus- tai tuoteperheitä, joita varten lähinnä olioparadigman sisällä kehitettiin sovelluskehyksen käsite. (Koskimies & Mikkonen, 2005, 16.)

Arkkitehtuuritason käsitteiden ja toteutusmekanismien pohjalta lähtevää ohjelmistojen luomista tapahtuu jatkuvasti enemmän. Järjestelmien monimutkaistuminen ja kasvaminen sekä uudelleenkäytön merkityksen korostuminen yleisen arkkitehtuuritason ratkaisuja korostavan teknologiakehityksen ohella vaikuttavat arkkitehtuurilähtöisen kehityksen kasvuun. (Koskimies & Mikkonen, 2005, 16.)

2.1.1 Komponentit

Jo ennen ohjelmistoarkkitehtuurien esiintuloa on ollut puhetta ohjelmistokomponenteista, joita käytettäisiin kokoamaan sovelluksia samankaltaisesti kuin komponentteja käytetään elektronisten laitteiden kokoamiseen. Komponenttien käsite ohjelmistotekniikassa on kehittynyt erityisesti uusien komponenttiteknologioiden myötä ja nykyisin komponentilla on monissa yhteyksissä ollut konkreettinen ja täsmällinen merkitys. (Koskimies & Mikkonen, 2005, 53.)

Koskimies ja Mikkonen (2005, 53) määrittelevät ohjelmistokomponentin ”itsenäiseksi ohjelmistoyksiköksi, joka tarjoaa palvelujaan hyvin määriteltyjen rajapintojen kautta”.

2.1.2 Rajapinnat

Rajapinnan ajatuksena on erottaa toisistaan se, mitä halutaan saada aikaan ja miten se toteutetaan. Näin ollen palvelun käyttäjän ei tulisi olla riippuvainen mistään palvelun tietystä osasta, kuten toteuttajasta, toteutustavasta tai komponentista vaan ainoastaan palvelusta itsestään abstraktina käsitteenä. Rajapinta esittää tämän abstraktin käyttäjälle. (Koskimies & Mikkonen, 2005, 57—58.)

Rajapinnan tulisi kertoa kaikki se oleellinen tieto, jota käyttäjän tarvitsee palvelusta tietää. Minimissään rajapinta kuitenkin kertoo sen, miten palvelu otetaan käyttöön, eli palvelun nimen, parametrit (sekä niiden tyypit) sekä mahdollisen tuloksen tyypin. Edellä mainitulle minimimääritykselle käytetään nimeä kutsumuoto (’*signature*’). (Koskimies & Mikkonen, 2005, 58.)

Rajapinnan käsite kehittynyt vähitellen. Aluksi se lähti varhaisten ohjelmointikielten kutsumuotomäärittelyistä ja tämän jälkeen modulaaristen ohjelmointikielten tuodessa abstraktin tietotyypin käsitteen, jolla joukko tietorakenteita ja niihin liittyviä aliohjelmia pystyttiin kokoamaan yhden moduulirakenteen sisään. Olioparadigma toi abstraktin tyypin käsitteeseen lisää laajennettavuutta antamalla mahdollisuuden luokkien periyttämiseen. Kun myöhemmin tuli mahdolliseksi luoda abstrakti luokka, joka ei itsessään toteuttanut mitään, vaan ainoastaan perii toteuttavat luokat itseensä, alettiin olla jo hyvin lähellä nykyistä rajapinnan käsitettä. (Koskimies & Mikkonen, 2005, 58.)

Uudemmissa ohjelmointikielissä kuten Javassa ja C#:ssa ei enää käytetä moniperintää abstrakteissa luokissa. Tämän vuoksi niitä ei voida käyttää toteuttamaan usean luokan toteuttamia metodeita, vaan näissä kielissä käytetään omaa rajapinta syntaksia. Näissä tapauksissa moniperintä on sallittu ja ne voivat toteuttaa useita rajapintoja. (Koskimies & Mikkonen, 2005, 59.)

2.2 Web-API:t

Web-API:t ovat käytännössä verkkopalveluiden keino avata sovelluskehitys kolmansille osapuolille. Nykyisin lähes kaikki suuret verkkopalvelut tarjoavat oman API:nsa kolmansien osapuolien käyttöön, esimerkiksi Twitter, Flickr, del.icio.us, Google, Facebook ja Youtube. Mikrobloggauspalvelu Twitterin API on esimerkiksi saanut suurta suosiota kehittäjien joukossa; Twitterin API:a hyödyntäviä asiakasohjelmia löytyy tällä hetkellä todella monia.

Web-API:t rinnastetaan joskus WWW-sovelluspalveluihin (’Web Service’). Web-API voidaan miettiä sarjana WWW-sovelluspalveluita, jossa jokaisella on yksi tai useampia toimintoja, joita voidaan kutsua Internetin avulla. (Gosnell, 2005, 2.)

2.2.1 WWW-sovelluspalvelu (’*Web Service*’)

Web-standardien kehityksestä vastaavan organisaation W3C:n toimesta WWW-sovelluspalvelu on määritelty vuonna 2004 laaditussa ’Web Services Architecture’-dokumentissa seuraavanlaisesti: WWW-sovelluspalvelu on verkon yli toimiva koneiden välistä vuorovaikutusta tukeva ohjelmistojärjestelmä. Sillä on rajapinta, joka on kuvailtu koneellisesti käsiteltävässä muodossa (erityisesti WSDL). Muut järjestelmät ovat vuorovaikutuksessa WWW-sovelluspalveluun sen kuvauksessa määritetyllä tavalla

[1] “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

käyttäen SOAP-viestejä, tyypillisesti käyttäen HTTP:n ja XML:n yhdistelmää yhdessä muiden WWW-standardien kanssa[1]. (Web Services Architecture, 2004.)

Vaikka W3C määritteli termin vasta vuonna 2004, ovat WWW-sovelluspalvelut olleet konseptina käytössä Webin alkuajoista lähtien. Vuodesta 1994 asti useat organisaatiot ovat kehittäneet useilla eri nimillä jaettavia olioita, jotka tukivat WWW-sovelluspalveluita. NeXT käytti nimitystä *Portable Distributed Objects* (Siirrettävät Hajautetut Oliot), Microsoft käytti nimitystä *Component Object Model* (*COM*, Komponentti Olio Malli), IBM käytti nimitystä *System Object Model* (*SOM,* Järjestelmä Olio Malli) ja Apple nimitystä *OpenDoc* (Avoin Dokumentti). (Alesso & Smith, 2004, 122.)

Netscapen perustajajäsen Marc Andreessen kuvaili vuonna 1996 kuinka liiketoiminnassa voitaisiin hyödyntää Internet-protokollia palveluiden hakuun alustalta toiselle. Tunnettuihin palveluihin kuului *Business To Consumer* (liiketoiminnalta kuluttajalle) eli B2C-palvelut, kuten hakukoneet, osakelaskurit ja luottokorttipalvelut. B2C-palveluissa tarjottiin Webbiä rajapintana kun taas *Business-to-Business* (Liiketoiminnalta liiketoiminnalle) eli B2B palveluiden vuorovaikutuksessa käytettiin patentoituja ohjelmointirajapintoja. Uusia palveluita luotiin jatkuvasti lisää, jokaisen tavoitteena tuottaa liiketoimintamalli, joka tuottaisi entistä paremman ja kehittyneemmän palvelun. Yhteisenä päämääränä kaikille näille palveluille oli standardeihin perustava avoin rajapinta ohjelmiin Webin yli. (Alesso & Smith, 2004, 122.)

WWW-sovelluspalveluiden toiminta vaatii, että kehittäjillä on olemassa yhteisiä standardeja käytössään, ettei kehitys lukkiutuisi yhteen ympäristöön. Tarkemmin sanottuna tarvittiin tekstipohjaisia Internet-protokollia, joita olisi mahdollista käyttää millä tahansa alustalla tai kielellä. (Alesso & Smith, 2004, 123.)

Ennen Webbiä olisi voinut käytännössä olla mahdotonta saada kaikkia suuria ohjelmistotoimittajia sitoutumaan yhteen protokollaan verkon yli toimivaa ohjelmistotoimintaa varten. Web tarjosi kuitenkin ratkaisun täsmentämällä alemman tason tiedon siirtoa standardisoitua kommunikointia varten. Web perustuu HTTP-protokollan käyttöön TCP/IP-protokollan päällä. HTTP antoi vakaan pohjan WWW-sovelluspalveluille, sillä se oli jo olemassa oleva ja luonnollinen tapa siirtää ja linkittää tietoja. HTTP:ssa jokaista resurssia vastaa oma URI (*Uniform Resource Identifier,* Yhteinen Resurssi Tunniste).

HTTP:n avulla jokainen resurssi voidaan linkittää toiseensa, eikä ohjelmien välille luoda riippuvuuksia; on olemassa ainoastaan linkitettyjen resurssien ”Verkko” eli Web. (Alesso & Smith, 2004, 123.)

Kun Web ratkaisi tiedonsiirtovaatimukset, tarvittiin WWW-sovelluspalveluita varten vielä yhteinen viestintä ja tiedon kapselointi (*’encapsulation’*) standardi. XML oli luonnollinen valinta ohjelmien väliseen viestintään, sillä se oli alustariippumaton laajalti huomiota herättänyt standardi tiedon kuvaamiseen. HTTP-metodit pystyvät toimimaan minkä tahansa tietojen kanssa käyttäen XML-sanastoa. Tämä luo yhteensopivuusongelman, sillä keskenään eroavat XML-sanastot eivät automaattisesti toimi keskenään. Tätä varten on kuitenkin luotu työkaluja kuten XSLT, joka on XML-pohjainen merkintäkieli XML-tiedostojen muunnoksiin. Toinen tapa yhteensopivuusongelman ratkaisuun on yhteisten mallien standardisointi ja jakaminen erilaisten rekisterien, arkistojen ja standardisointi -runkojen avulla. (Alesso & Smith, 2004, 123–124.)

2.2.2 SOAP (*Simple Object Access Protocol*)

SOAP on W3C:n suosittelema kevyt protokolla rakenteellisen informaation vaihtoon hajautetussa ympäristössä (SOAP Version 1.2 Part 1: Messaging Framework, 2007).

SOAP:n tehtävänä WWW-sovelluspalveluissa (ja Web-API:ssa) on toimia standardisoituna paketointi-protokollana ohjelmien välisessä viestien jakamisessa. SOAP määrittelee XML-perustaisen kuoren informaatiolle ja asettaa säännöt kuinka ohjelmisto ja alusta-spesifit tietotyypit käännetään XML-muotoon. SOAP:n malli mahdollistaa sen käytön laajasti eri ohjelmistoviesteihin ja integrointimalleihin, mikä on osaltaan edesauttanut sen kasvavaan suosioon. (Snell, Tidwell & Kulchenko, 2001, 15.)

SOAP:n perustan muodostaa XML-viestintä, jossa ohjelmat jakavat tietoa keskenään XML-dokumenttien avulla. Dokumenttien viestinä voi olla mitä tahansa: ostotilaus, pyyntö osakkeiden hinnasta, kysely hakukoneelle, listaus vapaista lentolipuista tai mitä tahansa ohjelmaan liittyvä tieto tai kysely. Koska XML ei ole sidottu mihinkään tiettyyn ohjelmaan, käyttöjärjestelmään tai alustaan, voidaan XML-viestejä käyttää missä tahansa ympäristössä. Perustavana ideana on siis, että kaksi ohjelmaa riippumatta käyttöjär-

jestelmästä, ohjelmointikielestä tai mistään muusta teknisen toteutuksen yksityiskohdasta, voivat jakaa avoimesti tietoa keskenään käyttäen yksinkertaisia viestejä, joita molemmat ohjelmat ymmärtävät. SOAP määrittää standardin, kuinka muodostaa rakenteellisia XML-viestejä. (Snell ym., 2001, 15–16.)

XML-viestinnällä ja täten SOAP:lla on kaksi sukulaisohjelmaa: RPC sekä EDI. RPC eli *Remote Procedure Call* (Metodin Etä Kutsunta) on lähtökohta hajautetulle tietojenkäsittelylle. RPC:n avulla ohjelman metodi voi kutsua toista metodia välittäen argumentteja ja vastaanottaen palautusarvoja. EDI eli *Electronic Document Interchange* (Elektroninen Dokumenttien Vaihto) on lähtökohta automatisoidulle liiketoimintojen tapahtumille, määritellen taloudellisille ja kaupallisille dokumenteille ja viesteille standardin muodon ja tulkkauksen. Käytettäessä SOAP:ia EDI:ä varten (’*document-style*’ SOAP), XML-viestissä liikkuu ostostilaus, veronpalautus tai muu vastaava dokumentti. Käytettäessä SOAP:ia RPC:tä varten (’*RPC-style*’ SOAP) XML-viestissä liikkuu parametrin tai palautusarvon kuvaus. (Snell ym., 2001, 16.)

Kuten edellisen luvun (2.3.1) lopussa todettiin, eivät erilaiset XML-sanastot toimi automaattisesti yhteen, vaan tarvitaan työkaluja ja sopimuksia, joiden avulla ohjelmat toimivat. SOAP antaa tähän omat määrityksensä.

SOAP-viesti koostuu kuoresta, joka sisältää valinnaisen ylätunnisteen (’*header*’) sekä pakollisen rungon (’*body*’). Ylätunniste sisältää tietoja siitä, kuinka viesti tulisi käsitellä. Tiedot ovat lohkoissa, joissa voidaan kertoa tietoja esimerkiksi reititys- ja toimitusasetuksista, tunnistautumisesta, auktorisoinnista ja tapahtumien kontekstista. Viestin runko sisältää varsinaisen toimitettavan ja käsiteltävän viestin. Viestin runko voi sisältää mitä vain, mitä XML-syntaksilla voidaan ilmaista. XML-syntaksi SOAP-viestiä varten perustuu *http://www.w3.org/2001/12/soap-envelope* nimiavaruuteen. SOAP-kuoren rakenne näkyy kuvassa 5. (Snell ym., 2001, 17.)

KUVA 5. SOAP-viestin rakenne

SOAP:ia käyttäviä WWW-sovelluspalveluita (ja API:ja) kehitetään yleisesti työkalupakkien avulla. Useimpien kehittäjien ei edes tarvitse tietää kaikkia yksityiskohtia, esimerkiksi siitä kuinka SOAP-viestien kuori muodostuu tai miten tiedot muunnetaan XML-muotoon. Kehittämistä varten valitaan vain itselle sopiva työkalupakki tai hyödynnetään kielistä valmiiksi löytyviä ominaisuuksia. (Snell ym., 2001, 39.)

Monet eri kielet tarjoavat valmiita ominaisuuksia SOAP:n kehittämiseen palvelimen puolella, esimerkiksi PHP:sta löytyy luokka nimeltä SoapServer (PHP: SoapServer – Manual. 2010).

SOAP:ia käyttäviä WWW-sovelluspalveluita (ja API:ja) kutsutaan hyödyntäen HTTP:n GET- ja POST –metodeita. Kutsua varten muodostetaan dokumentti edellä kuvatun SOAP-viestin rakenteen mukaisesti. Katso esimerkkidokumentti kuvasta 6. (Gosnell, 2005, 5.)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:SOAP-ENC=”http://schemas.xmlsoap.org/soap/encoding/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
SOAPENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Body> <m:getUserInfo xmlns:m=”http://arcweb.esri.com/v2”> <token
xsi:type=”xsd:string”>MyToken</token> </m:getUserInfo>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

KUVA 6. Dokumentti SOAP-kutsua varten. (Gosnell, 2005)

Monista kielistä löytyy valmiita työkaluja SOAP-viestien muodostukseen, joten ei ole välttämättä tarpeellista osata itse muodostaa SOAP-viestejä puhtaalta pöydältä. Esimerkiksi .NET Framework osaa muodostaa valmiita SOAP-viestejä ja myös PHP:sta on versiosta 5 lähtien löytynyt oma SoapClient-luokka SOAP:n hyödyntämistä varten.

SOAP:iin liittyy myös tärkeänä osana WSDL, eli *Web Service Description Language* (WWW-sovelluspalveluiden Kuvaus –Kieli). WSDL on XML-pohjainen kieli, jonka kehityksen Microsoft ja IBM aloittivat. WSDL:n tarkoitus on toimia dokumenttina, joka kertoo mitä viestejä WWW-sovelluspalvelu ottaa vastaan ja mitä se tuottaa vastauksena. (Gosnell, 2005, 6)

2.2.3 REST (*Representational State Transfer*)

Nykyiset WWW-sovelluspalvelu-arkkitehtuurit keksivät uudelleen tai sivuuttavat kaikki ne piirteet, jotka tekevät Webistä suositun, väittävät Richardson ja Ruby (2007, xiii).

Richardson ja Ruby muistuttavat, että jokainen verkkosivu on yhtälailla WWW-sovelluspalvelu. He nostavat esimerkkinä Googlen hakupalvelun, joka on verkkosivu, mutta samalla etänä toimiva WWW-sovelluspalvelu, joka käy läpi massiivisia tietokantoja ja palauttaa muotoillun vastauksen. (Richardson & Ruby, 2007, xiii.)

Samojen piirteiden, jotka tekevät verkkosivuista helppokäyttöisiä verkon käyttäjille, pitäisi tehdä WWW-sovelluspalveluista helppokäyttöisiä ohjelmoijille. REST eli *Representational State Transfer* (Tilan Esityksen Siirto) pyrkii tuomaan näitä piirteitä esiin perustason Web –tekniikoiden avulla, hyödyntäen protokollia: HTTP, URI ja XML. (Richardson & Ruby, 2007, xiii.)

Richardson ja Ruby uskovat, että WWW-sovelluspalveluita on mahdollista luoda ilman raskaita standardeja, ainoastaan hyödyntäen edellä mainittuja perustason Web-tekniikoita. Toisinaan joitain raskaampia ja monimutkaisempia protokollia voidaan hyödyntää, kuten aiemmin esiteltyä SOAP:ia toteuttamaan RPC-ohjelmia. (Richardson & Ruby, 2007, xiii.)

Monimutkaisten ja raskaiden protokollien etuna on, että modernit työkalut mahdollistavat WWW-sovelluspalveluiden luonnin yhdellä klikkauksella. Javalla tai C#:lla työskennellessä RPC-tyylisen WWW-sovelluspalvelun luonti onnistuu suoraan käyttäen valmiita työkalupakkeja. Nämä kätkevät kaiken monimutkaisuuden taakseen, mutta vähentävät tehokkuutta valtavalla XML-kuormalla. Käytännössä kuitenkin REST-tyylinen WWW-sovelluspalvelu olisi paljon yksinkertaisempi ja tehokkaampi. (Richardson & Ruby, 2007, xiii.)

Richardson ja Ruby määrittävät REST-tyyliset WWW-sovelluspalvelut palveluiksi, jotka ”näyttävät” Webiltä. He kutsuvat näitä Webiltä näyttäviä REST-tyylisiä palveluita resurssi orientoituneiksi (’*resource-oriented*’). (Richardson & Ruby, 2007, 13.)

REST-tyylisessä arkkitehtuurissa metodin tiedot liikkuvat HTTP-metodeissa. Resurssi-orientoituneessa arkkitehtuurissa rajaustiedot (’*scoping information*’) liikkuvat URI:ssa. Useita pyyntöjä voidaan siis suorittaa pelkästään yhdellä HTTP-rivillä. Nyrkkisääntönä voidaan siis pitää, että mikäli metodin tiedot eivät täsmää HTTP-metodiin, ei palvelu ole REST-tyylinen ja mikäli rajaustiedot eivät ole URI:ssa, palvelu ei ole resurssi-orientoitunut. Nämä ovat myös vaatimuksia kyseisten arkkitehtuurien vaatimuksia. (Richardson & Ruby, 2007, 13.)

Webistä löytyy useita tunnettuja REST-tyylisiä WWW-sovelluspalveluita. Tällaisia ovat esimerkiksi palvelut jotka tuottavat Atomia tai sen muunnelmia kuten GDataa. Amazonin *Simple Storage Service* (S3) ja useimmat vain luku WWW-sovelluspalvelut jotka eivät käytä SOAP:ia ovat myös REST-tyylisiä. Myös staattiset verkkosivut sekä monet muut Web-sovellukset, kuten hakukoneet ovat REST-tyylisiä. (Richardson & Ruby, 2007, 13.)

REST-tyylistä WWW-sovelluspalvelua (tai API:a) kutsutaan käyttämällä lähes jokaisesta modernista ohjelmointikielestä löytyvää HTTP-kirjastoa. Jotta voitaisiin rakentaa kokonaisvaltainen WWW-sovelluspalvelun asiakasohjelma, täytyy HTTP-kirjaston tukea seuraavia asioita:

- HTTPS ja SSL-sertifikaatin validointi. Monet WWW-sovelluspalvelut eivät hyväksy HTTP-pyyntöjä lainkaan.
- HTTP:n metodeita GET, HEAD, POST, PUT ja DELETE. Jotkin kirjastot tukevat vain GET- ja POST –metodeita, toiset on suunniteltu yksinkertaisiksi ja tukevat ainoastaan GET -metodia. Kaikkein käytetyimpiä ja tarpeellisimpia ovat GET ja POST, sillä esimerkiksi HTML-lomakkeet tukevat ainoastaan näitä metodeita.
- Mahdollisuus muokata PUT- ja POST –pyynnöissä lähetettäviä runkoja.
- Mahdollisuus muokata HTTP:n ylätunnisteita (’*headers*’).
- Pääsy HTTP-vastikkeen vastikekoodiin (’*response code*’) ja ylätunnisteisiin, ei ainoastaan HTTP-vastikkeen runko-osaan.
- Mahdollisuus kommunikoida HTTP:n välityspalvelimien kautta. Monet asiakasohjelmat yritysympäristöissä pystyvät toimimaan ainoastaan välityspalvelimien kautta, joten tämä on tärkeä, vaikkakin helposti unohdettava vaatimus.

(Richardson & Ruby, 2007, 29–30.)

Esimerkkeinä HTTP-kirjastoista ovat Pythonissa httplib2, Javassa HttpClient ja PHP:ssa libcurl (Richardson & Ruby, 2007, 33, 34, 37).

3 OHJELMOINTIRAJAPINNAN TOTEUTUS

3.1 Peloton API:n määrittäminen

3.1.1 Metodien määrittäminen

Ensimmäinen tehtävä API:n toteuttamisessa on tunnistaa liiketoiminnan kannalta oleellisia toimintoja, joita API:n pitäisi tarjota. On tärkeää tehdä itselleen selväksi, mitä ollaan tekemässä ja mihin API:n toteutuksella pyritään, ennen kuin aloitetaan varsinainen toteuttaminen. Tähän liittyy myös tarjottavien toimintojen määrittäminen. (Gosnell, 2005, 213.)

Peloton jakautuu kolmeen osa-alueeseen, kuten johdannossa kerrottiin. Alkuperäisenä ideana oli toteuttaa mobiiliasiakasohjelma, jolla saataisiin harjoitusohjelmaosiota käytettyä mobiililaitteella. Harjoitusohjelma on myös maksullinen osio, joten on tärkeää pyrkiä tuottamaan lisäarvoa tämän osion käyttäjille. Tämän vuoksi harjoitusohjelmaosiota pidettiin ensisijaisena kohteena API:a suunniteltaessa.

Harjoitusohjelman ideana on käytännössä tuottaa harjoitusohjelma viikon jokaiselle päivälle ja mahdollisuus pitää kirjaa suoritetuista harjoitteista. Oleellisin toiminto harjoitusohjelmassa on harjoitusohjelmien haku, sillä käyttäjät maksavat heille tehdyistä harjoitusohjelmista. Tästä syystä otimme API:n toteutukseen mukaan harjoitusohjelmien noutoon liittyviä toimintoja. Toteutuksesta rajattiin pois harjoitteiden kirjaamisen, koska tätä toimintoa ei pidetty ensiarvoisen tärkeinä. Lisäksi näin ensimmäinen versio API:sta pysyy tietoturvallisena, koska tietokannan suuntaan ei sallita kuin hakutoiminnot.

Harjoitusohjelmat toteutetaan viikko kerrallaan, joten pidettiin tärkeänä ottaa mukaan API:iin viikon harjoitusohjelman noutamisen. Tämän lisäksi oli tärkeää pystyä noutamaan API:sta meneillään olevan päivän harjoitusohjelma. API:n toteutukseen rajattiin siis seuraavat toiminnot:

- Viikon harjoitusohjelman nouto
- Päivän harjoitusohjelman nouto

3.1.2 Protokollien määrittäminen

Kun tarjottavat toiminnot on päätetty, täytyy päättää mitä protokollia API tulee tukemaan. Jotkin API:t voivat tukea useampia kuin yhtä protokollaa; esimerkiksi Amazon.com tukee sekä SOAP:ia että REST:iä. (Gosnell, 2005, 214.)

Vaihtoehtoisina toteutustapoina pidettiin joko REST-tyylistä tai SOAP-tyylistä rajapintaa. Mahdollisuutena olisi ollut myös näiden sekoitus tai toteuttaa tuki molemmille protokollille. Tuki molemmille protokollille pudotettiin pois liian työläänä vaihtoehtona.

Tarandeep Singh listaa verkkoartikkelissaan (2009) REST:n ja SOAP:n etuja seuraavasti:

SOAP

- Useimmiten helppo käyttää
- Noudattaa tiukkaa tyypitystä ja sopimuksia
- Paljon valmiita työkalupakkeja

REST

- Kevyt, eikä sisällä ylimääräisiä XML-merkintöjä
- Helposti ymmärrettävät tulokset
- Helppoja toteuttaa. Ei tarvita ylimääräisiä työkaluja

SOAP:n eduksi voidaan laskea valmiit työkalupakit, joiden avulla API:n toteuttaminen tehdään erittäin helpoksi; valitset haluamasi toiminnot, jotka haluat tarjota API:ssa ja työkalut hoitavat työn puolestasi. Täten kehittäjän ei välttämättä tarvitse tietää kaikkea mitä käytännössä tapahtuu. (Snell ym., 2001, 39.)

Richardson & Ruby kuitenkin tyrmäävät SOAP:n työkalujen tuomat edut. Heidän mielestään nämä työkalut voivat kätkeä monimutkaisuuden, mutta eivät silti oikeuta sitä, vaan yleensä jopa lisäävät tuota monimutkaisuutta. SOAP:n etuna pidettävä todella

tarkka tyypitys ja työkalujen avulla kehittäminen ovat siis samalla myös SOAP:n haittoina pidettäviä asioita, sillä ne lisäävät ylimääräistä monimutkaisuutta ja tekevät toteutuksesta näin raskaamman. (Richardson & Ruby, 2007, xvi.)

Yhtenä ajatuksena API:n toteuttamisen takana oli saada palvelua tarjottua yhä useammille päätelaitteille, kuten esimerkiksi mobiililaitteille. SOAP on tarkkaan tyypitetty ja SOAP-tyylisessä rajapinnassa yleensä liikkuu paljon tarkentavaa tietoa ja tämän vuoksi sen XML-liikenne on REST-tyylistä toteutusta raskaampaa. SOAP-tyylisen rajapinnan hyödyntämistä varten yleensä tarvitaan raskaampia työkaluja, kun taas REST-tyylisen rajapinnan tulosteiden hyödyntämiseen riittää yksinkertainen XML-parseri. Tämän vuoksi Pelottoman API:ssa päädyimme käyttämään REST-tyylistä toteutusta.

3.1.3 Maksullisuus ja salaaminen

Gosnell mainitsee yhdeksi mietittäväksi asiaksi maksullisuuden miettimisen (2005, 215). Koska API:n tarjoamat toiminnot liittyvät harjoitusohjelmaan, joka on maksullinen osio, ovat API:n toiminnot teoriassa maksullisia. Tosin maksullisuus ei kohdistu API:n käyttämiseen vaan harjoitusohjelman käyttöön.

Tämä tarkoittaa Pelottoman API:n kohdalla sitä, että harjoitusohjelman haku onnistuu ainoastaan siten, että käyttäjä tunnistautuu palveluun ja mikäli käyttäjällä on oikeus harjoitusohjelmaan (on siis maksanut palvelusta), annetaan harjoitusohjelma vastauksena rajapinnasta.

Käyttäjän tunnistautumiseen palataan tarkemmin luvussa 3.5. Rajapintaa laajennettaessa voitaisiin tarjota myös tietoa esimerkiksi Pelottoman järjestämistä tapahtumista, jolloin tiedon ei tarvitsisi olla salattua tai käyttäjätunnuksen taakse piilotettua.

3.2 Peloton.fi tekniikka

Peloton-verkkopalvelu on toteutettu WordPress –julkaisualustan päälle. ”WordPress on moderni henkilökohtainen julkaisualusta. Sen painopisteinä on esteettisyys, web-

standardit ja käytettävyys.” kertoo WordPress|Suomi –sivusto (2010). WordPress on avoimen lähdekoodin projekti, joten lähdekoodi on vapaasti käytettävissä ja muokattavissa.

WordPressiä koskevissa teknisissä asioissa apua haettiin WordPress Codex -sivustolta. WordPress Codex –sivustolta löytyy dokumentaatiota WordPressistä sekä ohjeistusta WordPressin kanssa toimimiseen.

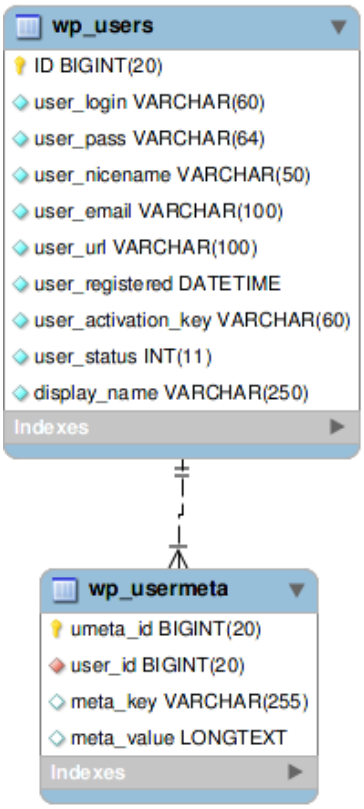
Toteutettavassa rajapinnassa käytettiin mahdollisimman paljon hyväksi WordPressin tarjoamaa valmista arkkitehtuuria ja valmiita metodeja. Lisäksi käytettiin Pelottoman luomaa valmista tekniikkaa hyväksi. Tarkoituksena on kuitenkin, että alkuperäiseen koodiin ei tehdä muutoksia ohjelmointirajapintaa luotaessa.

3.2.1 Tietokannat

WordPress käyttää tietojen ja asetuksien säilyttämiseen MySQL-tietokantaa. WordPressin puhtaassa asennuksessa luodaan 11 tietokannan taulua, joissa säilytetään esimerkiksi käyttäjien luomaa sisältöä, kommentteja, metatietoja ja käyttäjätilejä. Tämän opinnäytetyön kannalta oleellisia tauluja näistä ovat wp_users- ja wp_usermeta- taulut. Näiden taulujen välillä on isä-lapsi-yhteys, jossa isätauluna (’*master table*’) on wp_users ja lapsitauluna (’*reference table*’) wp_usermeta.

Isä-lapsi-yhteydeksi nimetään sellaisia taulujen välisiä yhteyksiä, joissa isätaulusta löytyvälle riville voi löytyä useita rivejä lapsitaulusta. Nämä rivit linkittyvät toisiinsa perusavaimen (’*primary key*’ tai PK) ja viiteavaimen (’*foreign key*’) avulla. Lapsitaulusta löytyvällä viiteavaimella linkitetään lapsitaulun rivit isätaulun riviin. (Hovi, Huotari & Lahdenmäki, 2003, 9.)

Tässä tapauksessa wp_users-taulussa löytyvä ID toimii isätaulun perusavaimena. Lapsitaulussa wp_usermeta löytyvä user_id toimii viiteavaimena ja viittaa isätaulun perusavaimeen. Kuvassa 7 on taulujen koko rakenne. WordPress Codex –sivustolla on nähtävissä koko tietokannan rakenne.



KUVA 7. Toteutuksen kannalta merkittävät taulut wp_users ja wp_usermeta (Database Description << WordPress Codex, 2010).

Näiden perustaulujen lisäksi Peloton-verkkopalvelusta löytyy muita tauluja, jotka on luotu palvelun tarpeiden mukaan. Tässä opinnäytetyössä tarvitaan äsken esiteltyjen käyttäjätietotaulujen lisäksi taulua nimeltään wp_bp_trainer_week_program, joka sisältää tiedot harjoitusohjelmista.

3.3 Toteutustekniikat

PHP eli PHP Hypertext Preprocessor on palvelinpuolen ohjelmointiin tarkoitettu skripti-kieli. PHP esiintyy usein suositussa yhdistelmässä: Linux + Apache + PHP + MySQL, jota käytetään todella paljon WWW-puolella. PHP:ssa ohjelmointi tapahtuu käsitteillä, jotka ovat ihmisille luonteenomaisia ja tuttuja, tästä johtuen PHP:sta käytetään ilmaisua ”korkean tason kieli”. (Kolehmainen, 2006, 3.)

PHP oli luonnollinen valinta ohjelmointirajapinnan toteutukseen, koska Peloton-verkkopalvelu on toteutettu sillä. Käyttämällä samaa ohjelmointikieltä, on mahdollista

ottaa valmiiksi Peloton-verkkopalvelusta löytyviä funktioita ja asetuksia käyttöön rajapinnan toteuttamisessa.

MySQL on maailman suosituin avoimen lähdekoodin tietokantaohjelmisto (MySQL, 2010). Kuten mainittua, MySQL esiintyy yhdessä PHP:n kanssa useasti. Peloton-verkkopalvelun pohjana toimii SQL-tietokanta, jonka hallintajärjestelmänä on MySQL, joten sitä käytetään myös rajapinnan toteutuksessa. Tietokantaan ollaan yhteydessä PHP:sta käsin tehtävillä SQL-hauilla.

Atom Publishing Protocol on XML-sanasto aikaleimattuja merkintöjä sisältävien listojen kuvaamiseen. Atomista löytyy paljon valmiita tageja julkaisemista varten. Atom on käytännöllinen XML-sanasto, sillä usein WWW-sovelluspalvelut (ja siis myös API:t) ovat tiedon julkaisemisen kanavia. Lisäksi nykyään on valmiiksi monia WWW-sovelluspalveluiden asiakasohjelmia, jotka ymmärtävät Atomia valmiiksi. Tämä mahdollistaa API:n nopean käyttöönoton. (Richardson & Ruby, 2007, 263.)

Atomista on tehty myös standardiesitys vuonna 2005 (Nottingham & Sayre, 2005). Kuvassa 8 on esimerkki Pelottoman rajapinnasta saatavasta Atom –syötteestä.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <id>http://www.peloton.fi/api/get_days_training.php</id>
    <title>Päivän harjoitusohjelma</title>
    <updated>2010-11-04T00:00:00+02:00</updated>
    <link rel="self" href="http://www.peloton.fi/api/get_days_training
.php" />
    <category term="kuntotaso">03</category>
    <category term="urheilulaji">Juoksu</category>

    <rights>© 2010 Peloton Liikuntakerho Oy</rights>
    <author>
        <name>Peloton Liikuntakerho Oy</name>
        <uri>http://www.peloton.fi/</uri>
    </author>
    <entry>
        <id>http://www.peloton.fi/api/get_training.php?year=2010&week_
nro=&train_nro=1</id>

        <title>Kävely/Hölkkä 1h</title>
        <updated>2010-11-04T00:00:00+02:00</updated>
        <con-
tent>Tee noin tunnin mittainen lenkki matalalla sykkeellä. Ainakin ylä
mäet voit kävellä, jos syke tuntuu niissä nousevan herkästi. </content
>
        <category term="paiva" label="" />
    </entry>

</feed>
```

KUVA 8: Atom –syöte päivän kunto-ohjelmasta.

Rajapinnan kutsuosoitteiden muokkaamiseen käytetään .htaccess-tiedoston muokkaamista. Tämä tiedosto antaa pääsyn palvelimen HTTP-asetuksiin. Lisäksi .htaccess antaa mahdollisuuden käsitellä virheellisiä rajapintakutsuja, jolloin pystytään palauttamaan hallittu virheraportti. Alla esimerkki .htaccess-tiedoston käytöstä API:ssa (kuva 9). (Apache Tutorial: .htaccess files, 2010.)

```
RewriteEngine on

#viikon treenin uudelleen ohjaus
RewriteRule ^api/training/week$ /training/week/
RewriteRule ^api/training/week/$ /get_weeks_training.php

#Päivän treenin uudelleen ohjaus
RewriteRule ^api/training/day$ /training/day/
RewriteRule ^api/training/day/$ /get_days_training.php

#Virheellinen osoite, niin tarjotaan virhe xml
ErrorDocument 404 /404error.xml
```

KUVA 9: .htaccess-tiedoston sisältöä

3.4 API:n arkkitehtuuri

Rajapinnan toteutus voidaan jakaa mielekkäästi kahteen osaan. Ensimmäinen osio kattaa rajapinnan toiminnot tarjoava luokka, jota kutsumme Pelottoman tapauksessa nimellä PelotonApi. Toinen osio kattaa tunnistautumisen ja varsinaisen rajapintatoiminnon.

Aluksi suunniteltiin rajapinnan toiminnot toteuttava luokka. Ajatuksena oli, että API:n tarjoamat metodit löytyvät luokasta julkisina metodeina. Tämän jälkeen luokan sisällä jaettiin tehtäviä pienempiin yksityisiin metodeihin, jotta luokan koodi pysyisi mahdollisimman helppolukuisena. Kuvassa 10 on PelotonApi-luokan luokkakaavion.

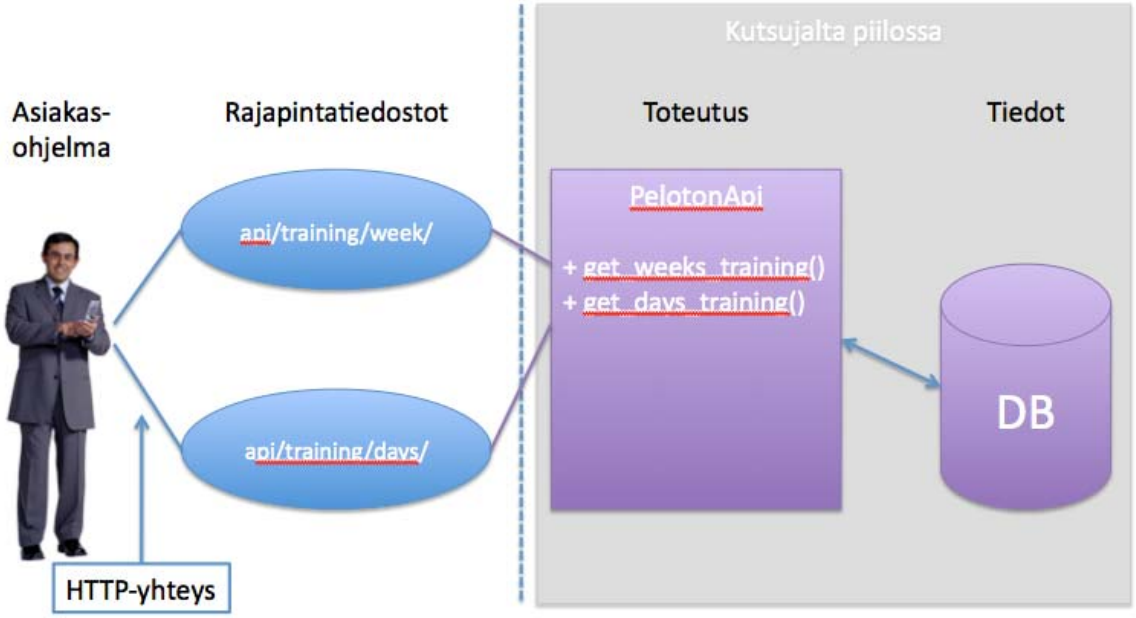
KUVA 10. PelotonApi:n luokkakaavio.

PelotonApi-luokkaan periytetään myös virheentunnistus luokka nimeltään ApiDebug, jotta rajapinnan toimintaa voidaan tarkkailla ja mahdollisia virheitä etsiä. Virheentunnistusta ei kuitenkaan tässä opinnäytetyössä tarkastella tämän enempää.

PHP-luokka on helppo ottaa käyttöön palvelimen puolella, mutta palvelimen ulkopuolelta se ei onnistu. Ulkopuolisia kehittäjiä varten tarvitaan rajapintatiedostoja, jotka yhdistävät PHP-luokan ja kehittäjät. (Campbell, 2006.)
Ohjelmistokehityksessä on tärkeää pitää erillään haluttu tulos ja ne keinot, kuinka tähän tulokseen pyritään pääsemään. Halutut tulokset tulee tarjoilla käyttäjälle abstraktion tasolla ilman, että käyttäjä joutuu olemaan riippuvainen esimerkiksi toteutustekniikasta tai ohjelmointikielestä. (Koskimies & Mikkonen, 2005, 57.)

Toteutusta varten luotavat rajapintatiedostot tarjoavat kutsujilleen Atom –syötteen oikein kutsuttaessa. Tällä tavoin voidaan rajapinta erottaa abstraktiotasolla omaksi yksikökseen. Nyt kutsuja ei ole sidottuna rajapinnan toteutuksen kieleen tai esimerkiksi tietokannassa tapahtuviin muutoksiin. Jos palvelun sisällä tehdään muutoksia, se ei vaikuta ulkopuoliseen kutsujaan, kunhan kutsuttaessa samat säännöt pätevät edelleen.

Kuvassa 11 havainnollistetaan Pelottoman API:n toteutusta on esitetty graafisesti. Asiakasohjelma ottaa HTTP:n avulla yhteyttä rajapintatiedostoihin, jotka hyödyntävät PelotonApi-luokkaa, joka noutaa tietokannasta haluttavat tiedot ja palauttaa ne eteenpäin muotoiltuna. Tämän jälkeen rajapintatiedosto palauttaa tiedot HTTP:n avulla takaisin asiakasohjelmalle.



KUVA 11. Pelottoman API:n toiminta graafisesti esitettynä.

3.5 Tunnistautuminen

Pelottoman API:ssa käyttäjien tunnistaminen on tärkeässä roolissa, sillä API:sta haettava tieto on käyttäjäkohtaista, eikä ilman käyttäjän tunnistamista voida antaa tällä hetkellä API:sta mitään tietoja.

Mahdollisuutena olisi ollut toteuttaa itse jokin tunnistautumisjärjestelmä rajapintaan, mutta tätä pidettiin liian työläänä vaihtoehtona, sillä oli olemassa valmiita järjestelmiä, joita pystyttiin hyödyntämään. Toisena mahdollisuutena olisi ollut käyttää pitkään käytössä ollutta HTTP:sta löytyvää Basic Auth ominaisuutta. Basic Authissa käyttäjien tiedot eivät kuitenkaan pysy turvattuina ja ne joudutaan luovuttamaan asiakasohjelman päässä, joten tämä ei ollut varteenotettava vaihtoehto. Tällä hetkellä suosittu järjestelmä API:en käyttämään tunnistautumiseen on OAuth, jota käyttävät esimerkiksi Twitter, GoogleAPI, Yahoo ja Flickr.

Asiakaspuolen kirjastoja OAuthille on olemassa jo melko runsaasti. Tämä oli myös yksi syy OAuthin valintaan tunnistautumista mietittäessä. Ajan tasalla oleva lista kaikista OAuth-kirjastoista löytyy osoitteesta http://oauth.net/code/.

3.5.2 OAuth

Tunnistautumiseen ja valtuuttamiseen käytettiin OAuth-protokollaa. OAuth on turvallinen tapa myöntää oikeuksia rajapintaa hyödyntäville ohjelmille. OAuthin perusajatus on, että rajapintaa käyttävät ohjelmat eivät saa käsiinsä käyttäjien salasanoja tai käyttäjätunnuksia vaan ainoastaan oikeuden käyttää tietoja. Tämä on OAuthin suurin etu verrattuna esimerkiksi HTTP:n Basic Authenticationiin (Hammer-Lahav, 2007).

OAuth-tapahtumissa on mukana kolme osapuolta: Palvelin, kehittäjä/kuluttaja ja käyttäjä (Hammer-Lahav, 2007). Palvelin on tässä tapauksessa Peloton.fi rajapinta, kehittäjä/kuluttaja on ulkopuolisen kehittäjän sovellus ja käyttäjä on ulkopuolisen kehittäjän sovellusta käyttävä henkilö.

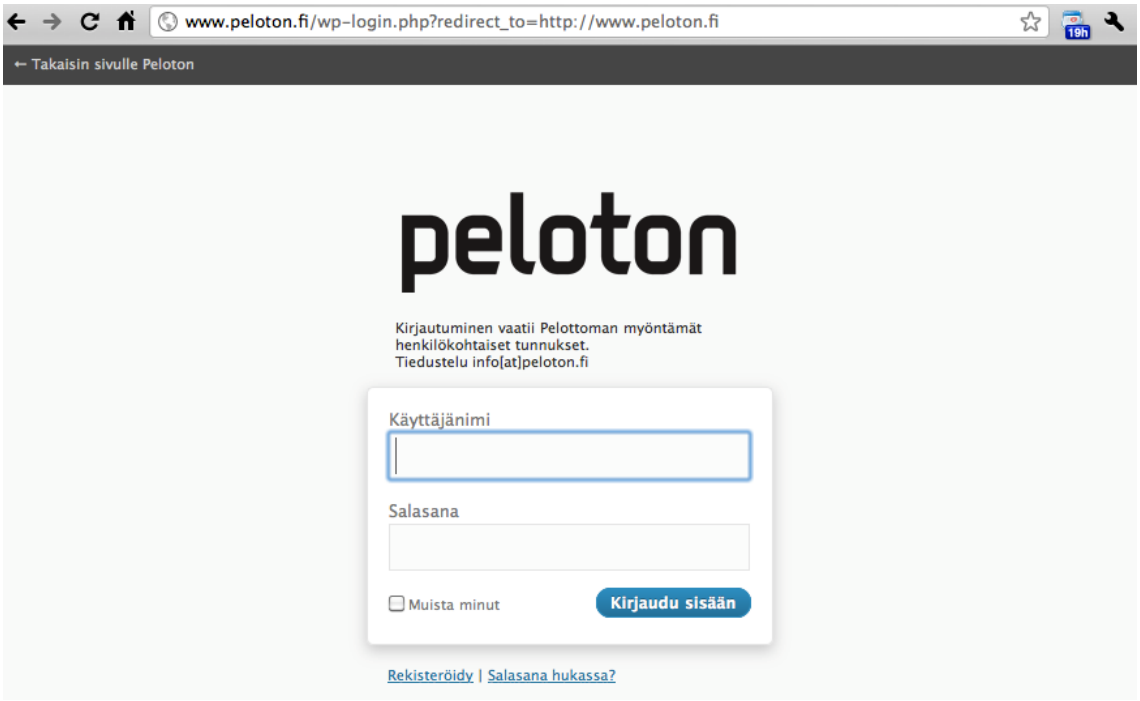
OAuthin ideana on, että jokainen rajapintaa käyttävä kehittäjä rekisteröityy rajapinnan käyttäjäksi ja hankkii palvelulta itselleen kaksi kehittäjän tunnistavaa merkkijonoa: kuluttaja-avain (’*consumer key*’) ja kuluttajasalaus (’*consumer secret*’). Näillä kahdella merkkijonolla ei vielä rajapinnasta saada mitään irti, vaan ne toimivat kehittäjän tunnistavana avaimena, jolla tunnistetaan kehittäjän lupa toimia rajapinnan kanssa. (Hammer-Lahav, 2007.)

Pelottoman rajapintaa käyttävän sovelluksen on ensin pyydettävältä käyttäjältä oikeus käyttää hänen tietojaan, ennen kuin rajapinnan käyttö on mahdollista. Tätä varten sovelluksessa on oltava toiminto, jolla käyttäjä voi kirjautua sisään, eli antaa oikeudet sovellukselle käyttää hänen tietojaan.

Käyttäjän kirjautuessa sisään, tapahtuu hänen tietämättään tärkeä vaihe tunnistautumisprosessissa. Sovellus lähettää rajapintaan pyynnön tunnistautumisesta joka sisältää kehittäjän saaman kuluttaja-avaimen ja kuluttajasalauksen. Vastauksena rajapinnasta so-

vellus saa pyyntöavaimen (’*request token*’), jota käytetään oikeuksien hakemiseen. Tärkeä huomio on, että tämä toiminto tapahtuu sovellusta ajavan käyttäjän siitä tietämättä. (Hammer-Lahav, 2007.)

Seuraavassa vaiheessa käyttäjä ohjataan Peloton.fi osoitteeseen antamaan oikeuksia ja tunnistautumaan palveluun. Ensin käyttäjä syöttää käyttäjätunnuksen ja salasanan palveluun (Kuva 12). Tämä tapahtuu WordPressin wp-login.php –tiedostossa.



KUVA 12: Käyttäjä tunnistautuu palveluun. Huomaa domain.

Käyttäjän klikattua kirjaudu sisään –painiketta, hänet ohjataan seuraavaan kohtaan, jossa häneltä kysytään, haluaako hän antaa oikeudet sovellukselle käyttää hänen tietojaan (Kuva 13). Tässä kohtaa tietoturvan kannalta tärkeä huomio on, että tämä kaikki tapahtuu palvelun tarjoajan palvelimen päässä, palvelun tarjoajan domainin alla ja että käyttäjän tunnuksia ei luovuteta missään vaiheessa asiakassovellukselle. (Hammer-Lahav, 2007.)

KUVA 13: Käyttäjältä kysytään lupa käyttää tietoja

PelotonApi-luokka tarvitsee toimiakseen käyttäjän ID-numeron. Kun edellisessä kohdassa käyttäjä kirjautui sisään omalle tililleen, voidaan käyttää WordPressin tarjoamaa funktiota wp_get_current_user(). Tämän funktio palauttaa WP_User –luokan ilmentymän, jonka yhtenä muuttujana on ID, eli käyttäjän ID-numero. Tämä ID-numero otetaan tunnistautumisen yhteydessä talteen.

Kun käyttäjä antaa oikeudet sovellukselle, ohjataan hänet takaisin asiakassovellukseen. Samalla kertaa kun käyttäjä ohjataan sovellukseen, lähetetään sovellukseen myös oikeutus-avain (’*Authorization key*’), joka on liitetty aiemmin haettuun käyttäjän ID-numeroon. Tämän jälkeen asiakassovellus lähettää oikeutus-avaimen peloton.fi palveluun, josta se saa käyttöönsä pääsytunnuksen (’*access token*’), jonka avulla se pystytään tunnistamaan API:ssa oikeudet antaneeksi käyttäjäksi. (Hammer-Lahav, 2007.)

OAuthin palvelimen puolen toteutukseen käytettiin valmista PHP-kirjastoa nimeltään oauth-php. Kirjasto tarjosi valmiit toiminnot, sekä valmiin esimerkin palvelimen puolen toteutuksesta.

Jokaisessa rajapintatiedostossa, johon ulkopuoliset ohjelmat ottavat yhteyttä, on käytetty OAuth-tunnistusta. Lisäksi on luotu neljä tiedostoa oauth-php –luokan ohjeiden mukaisesti, joiden avulla edellä kuvatulla tavalla saadaan annettua kolmannelle osapuolelle oikeudet rajapinnan käyttöön. Tiedostot ovat request_token.php, authorization.php, access_token.php sekä oauth_register.php. Näistä kolme ensimmäistä ovat mukana aina kun haetaan pääsytunnusta rajapinnan käyttöön. Neljännen tiedoston tehtävänä on luoda

kehittäjille aiemmin mainitut kuluttaja-avain ja kuluttajasalaus. Tiedostojen koodit löytyvät liitteenä (liite 1) opinnäytetyön lopusta.

Kehittäjiä varten tehdään oma sivunsa, josta he voivat hakea itselleen kuluttaja-avaimen ja kuluttajasalauksen. Tätä sivua varten tarvittava tekniikka löytyy oauth_register.php – tiedostosta ja löytyy opinnäytetyön liitteenä (liite 1). Tämä kyseinen sivu, sen grafiikka ja muu toteutus toteutetaan kuitenkin tämän opinnäytetyön ulkopuolella.

4 TULOKSIEN ESITTELY

4.1 Rajapinnan kutsuminen

Pelottoman API:ssa on kolme eri kutsuosoitetta, jonka jokaisen takaa löytyy yksi metodi. Alla näet kutsuosoitteet ja niiden takaa löytyvät toiminnot (taulukko 1). Metodit eivät ota mitään parametreja vastaan. Käyttäjän ID-numero otetaan talteen OAuth-auktorisoinnin yhteydessä.

TAULUKKO 1. Rajapinnan kutsuosoitteet eli metodit

Rajapinnan funktio	Kuvaus
api/training/week/	Palauttaa tunnistautuneen käyttäjän käynnissä olevan viikon treeniohjelman
api/training/day/	Palauttaa tunnistautuneen käyttäjän päivän treeniohjelman

Pelottoman API:a kutsuttaessa tulee käyttää OAuth-protokollan mukaista kirjattua kutsua. OAuth-kirjastoja löytyy verkosta valmiina ja monet kielet tukevat OAuthia suoraan.

4.2 Rajapinnan tulosteet

Kun API:iin tehdään kirjattu OAuth-kutsu, vastaa API Atom –syötteellä. Jokaisen Atom –syötteen voi jakaa kahteen osaan: syötteen tietoihin ja syötteen merkintöihin. Jokaisen vastauksen alussa kuvataan syötteen perustiedot. Otsikkona on haettava tieto, categorytageissa on mainittuna kuntotaso, sekä urheilulaji. Kuvasta 15 näkyy esimerkkinä, millainen vastauksen alku voi olla haettaessa viikon harjoitusohjelmaa.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<feed xmlns="http://www.w3.org/2005/Atom">
        <id>http://www.peloton.fi/api/get_weeks_training.php</id>
        <title>Viikon harjoitusohjelma</title>
        <updated>2010-08-05T00:00:00+00:00</updated>
        <link rel="self" href="http://www.peloton.fi/api/get_weeks_training.php" />
        <category term="kuntotaso" label="04" />
        <category term="urheilulaji" label="Kävely, Sauvakävely,
Hölkkä" />
        <rights>© 2010 Peloton Liikuntakerho Oy</rights>
        <author>
                <name>Peloton Liikuntakerho Oy</name>
                <uri>http://www.peloton.fi/</uri>
        </author>
```

KUVA 15. Atom –syötteen alku osa haettaessa viikon harjoitusohjelmaa.

Syötteen alkuosan jälkeen alkavat vastauksen merkinnät, jotka sisältävät haettua informaatiota. Jokaisessa merkinnässä on yksilöivä id, otsikko, luontipäivämäärä, sisältö sekä category –tagin sisältä löytyvä viikonpäivä. Kuvasta 16 näkyy esimerkki merkinnästä. Tällä hetkellä API:sta tulevat merkinnät sisältävät aina yhden harjoituskerran kuvauksen.

```
        <entry>
                <id>http://www.peloton.fi/api/get_traininging.php?
year=2010&amp;amp;week_nro=47&amp;amp;train_nro=5</id>
                <title>Kävely 30min + uinti</title>
                <updated>2010-08-05T00:00:00+00:00</updated>
                <content>Kevyttä kävelyä tai pyöräilyä, minkä jälkeen
voit mennä rentoutumaan esim. uimaan tai vesijuoksemaan (uimahalli,
uimaranta tai vaikkapa mökillä). Viileä vesi palauttaa lihaksia ja
niveliä rasituksista.</content>
                <category term="paiva" label="Keskiviikko" />
        </entry>
```

KUVA 16. Yksittäinen merkintä Atom –syötteessä, joka sisältää yhden harjoituskerran kuvauksen.

5 LOPPUSANAT

5.1 Kehitettävää

Pelottoman API saatiin valmiiksi opinnäytetyön tekemisen aikana. API:n toteutus onnistui tavoitteiden mukaisesti. Nyt kun perustekniikka on luotu, voi API:a alkaa laajentamaan lisäämällä metodeita PelotonApi-luokkaa ja luomalla vastaavan rajapintatiedoston.

Uusia metodeita rajapintaan tullaan lisäämään opinnäytetyön ulkopuolella. Seuraavat lisättävät metodit liittyvät edelleen harjoitusohjelmaan ja tarkemmin omien harjoituskertojen kirjaamiseen. Tämän lisäksi helposti toteutettavia metodeita voisivat olla esimerkiksi tapahtumakalenteriin liittyvät metodit, sillä ne eivät vaatisi minkäänlaista salausta tai tunnistautumista.

Kehitettävänä kohtana täytyy myös pitää API:n dokumentaatiota asiakasohjelmien kehittäjille. Opinnäytetyön aikana ei saatu luotua riittävän kattavaa ja helppoa ohjeistusta siitä, kuinka API:a pystyy hyödyntämään. Olisi varmasti myös hyödyllistä luoda valmiita luokkia, joiden avulla API:n käyttäminen olisi mahdollista vieläkin helpommin.

5.2 Hyödyllisyys

Suoraa hyötyä toimeksiantajalle on PelotonApi-luokasta, jota Pelottoman palvelimelta on mahdollista käyttää toteuttamaan esimerkiksi kevyempiä toteutuksia harjoitusohjelmasta.

API:n todellinen hyödyllisyys voidaan todeta vasta kun se julkaistaan. Uhkana on, että palvelun käyttäjistä ei löydy innokkaita kehittäjiä jolloin API jää hyödyttömäksi. Pelkona on myös, että kehittäjät kokevat API:n käytön liian vaikeaksi, jolloin API:n käyttö rajoittuu vain sisäiseen käyttöön.

5.3 Yhteenveto

Opinnäytetyön toimeksianto oli haastava ja se vaati paljon aiheeseen tutustumista. Verkkosivuille luotavat ohjelmointirajapinnat ovat kuitenkin selkeästi nousussa oleva trendi ja tämän takia niiden toiminnan hahmottaminen on erinomainen taito. WWW-sovelluspalveluiden kehityksestä on ollut keskustelua asiantuntijoiden kirjoituksissa ja näyttäisi siltä, että suurin osa niistä on REST:n puolella. Aika näyttää, mihin suuntaan todellisuudessa WWW-sovelluspalveluiden kehitys on menossa.

LÄHTEET

Kirjallisuus:

Alessa, H. & Smith, F. 2004. Developing Semantic Web Services. Wellesey: A K Peters, Ltd.

Campbell, R. 2006. How to Add an API to your Web Service. Luettu 13.9.2010. http://particletree.com/features/how-to-add-an-api-to-your-web-service/

Gosnell, D. 2005. Professional Web APIs: Google, eBay, Amazon.com, MapPoint, FedEx. Indianapolis: Wiley Publishing.

Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y. & Neyama, R. 2002. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Indianapolis: Sams Publishing.

Hovi, A., Huotari, J & Lahdenmäki, T. 2003. Tietokantojen suunnittelu & indeksointi. Jyväskylä: Docendo Finland Oy.

Kolehmainen, K. 2006. PHP & MySQL – Teoriasta Käytäntöön. Helsinki: Readme.fi.

Koskimies, K. & Mikkonen, T. 2005. Ohjelmisto-arkkitehtuurit. Helsinki: Talentum.

Richardson, L. & Ruby, S. 2007. RESTful Web Services. Sebastopol: O'Reilly Media, Inc.

Tidwell, D., Snell, J. & Kulchenko, P. 2001. Programming Web Services with SOAP. O'Reilly.

Verkkolähteet:

Apache Tutorial: .htaccess files. 2010. Luettu 8.10.2010. http://httpd.apache.org/docs/2.0/howto/htaccess.html

Hammer-Lahav, E. 2007. Beginners Guide to OAuth. Julkaistu 4.10.2007. Päivitetty 30.9.2010. http://hueniverse.com/oauth/

MySQL: About MySQL. Luettu 6.10.2010. http://www.mysql.com/about/

Nottingham, M. & Sayre, R. 2005. The Atom Syndication Format. [Network Working Group]. Julkaistu 12.2005. http://tools.ietf.org/html/rfc4287

Sanastokeskus TSK. 2005. TEPA-termipankki. Luettu 31.9.2010. [Termipankki]. http://www.tsk.fi/tepa

Senthilnathan, N. 2004. How to design Good APIs and Why they Matter. Luettu 14.9.2010. http://www.cincomsmalltalk.com/blog/blogView?showComments=true&entry=3258158706

Singh, T. 2009. REST vs. SOAP - The Right WebService. Luettu 13.9.2010. http://www.taranfx.com/rest-vs-soap-using-http-choosing-the-right-webservice-protocol

W3C Working Group. 2004. Web Services Architecture. [W3C Working Group Note]. Julkaistu 11.2.2004. Luettu 2.12.2010. http://www.w3.org/TR/ws-arch/

W3C. 2007. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). [W3C Recommendation]. Julkaistu 27.4.2007. http://www.w3.org/TR/soap12-part1/

WordPress | Suomi. Luettu 8.10.2010. http://fi.wordpress.org/

LIITTEET

LIITE 1

request_token.php

```
<?php
session_start();
require_once('printable/inc/config.php');
require_once('printable/inc/db.php');
/*
 * Initialize OAuth store
 */
require_once 'oauth-php/library/OAuthServer.php';
require_once 'oauth-php/library/OAuthStore.php';
require_once 'oauth-php/library/OAuthRequestVerifier.php';

$conn = new DB_connection();
OAuthStore::instance('MySQL', array('conn' => $conn->connection));

$server = new OAuthServer();
$token = $server->requestToken();
exit();
?>
```

authorization.php

```
<?php
session_start();
require_once('../wp-load.php');
require_once('api.php');
require_once('../wp-includes/pluggable.php');
require_once('../wp-includes/user.php');

require_once('printable/inc/config.php');
require_once('printable/inc/db.php');
/*
 * Initialize OAuth store
 */
require_once('oauth-php/library/OAuthServer.php');
require_once 'oauth-php/library/OAuthStore.php';
require_once 'oauth-php/library/OAuthRequestVerifier.php';

$conn = new DB_connection();
// Register the consumer
$store = OAuthStore::instance('MySQL', array('conn' => $conn-
>connection));

$server = new OAuthServer();
$current_user = wp_get_current_user();

// The current user
$user_id = $current_user->ID;
```

jatkuu

```
// Fetch the oauth store and the oauth server.
if($user_id == 0)
{
    $help_i = 0;
    foreach($_GET as $key => $value)
    {
        if($help_i >= 1)
        {
            $getit .= "&";
        }
        $getit .= $key . "=" . $value;
        $help_i++;
    }

    $url = "apiOauth/authorize.php?" . $getit;
    $url = urlencode($url);
header('Location: ../wp-login.php?redirect_to=' . $url);
}else{
try
{
    // Check if there is a valid request token in the current request
    // Returns an array with the consumer key, consumer secret, token,
 token secret and token type.
    $rs = $server->authorizeVerify();
    if ($_POST[allow])
    {
        // See if the user clicked the 'allow' submit button (or whate
ver you choose)
        $authorized = array_key_exists('allow', $_POST);

        // Set the request token to be authorized or not authorized
        // When there was a oauth_callback then this will redirect to
the consumer
        $server->authorizeFinish($authorized, $user_id);

        // No oauth_callback, show the user the result of the authoriz
ation
        // ** your code here **

   }else{
    $help_i = 0;
    $rest_url = false;

    $first ="";
    $rest = "";
    foreach($_GET as $key => $value)
    {
        if($help_i >= 1)
        {
            if($rest_url == true)
            {
                $rest .= "&";
            }else{
                $first .= "&";
            }
        }
        if($key == 'oauth_callback')
        {
            $first .= $key . "=";
            $rest .= $value;
            $rest_url = true;
        }else{
```

```
            if($rest_url == true)
            {
                $rest .= $key . "=" . $value;
            }else
            {
                $first .= $key . "=" . $value;
            }
        }
        $help_i++;
    }
    $url = "authorize.php?" . $first . urlencode($rest);
    echo get_auth_form();

   }
}
catch (OAuthException $e)
{
    // No token to be verified in the request, show a page where the u
ser can enter the token to be verified
    // **your code here**
    echo "Exception: " . $e;
}

}
?>
```

access_token.php

```
<?php
require_once('printable/inc/config.php');
require_once('printable/inc/db.php');
/*
 * Initialize OAuth store
 */
require_once('oauth-php/library/OAuthServer.php');
require_once 'oauth-php/library/OAuthStore.php';
require_once 'oauth-php/library/OAuthRequestVerifier.php';

$conn = new DB_connection();
// Register the consumer
$store =    OAuthStore::instance('MySQL', array('conn' => $conn-
>connection));

$server = new OAuthServer();
$server->accessToken();
?>
```

oauth_register.php

```
<?php
session_start();
require_once('../wp-load.php');
require_once('api.php');
require_once('../wp-includes/pluggable.php');
require_once('../wp-includes/user.php');
/*
 * Initialize OAuth store
 */
```

LIITE 1

```
require_once 'oauth-php/library/OAuthServer.php';
require_once 'oauth-php/library/OAuthStore.php';
require_once 'oauth-php/library/OAuthRequestVerifier.php';

$current_user = wp_get_current_user() ;

if($current_user->ID == 0)
{
    header("Location: ../wp-
login.php?redirect_to=apiOauth/oauth_register.php");
}else{
    // The currently logged on user
    $user_id = $current_user->ID;

    // This should come from a form filled in by the requesting user
    $consumer = array(
        // These two are required
        'requester_name' => $current_user->user_login,
        'requester_email' => $current_user->user_email,

        // These are all optional
        'call-
back_uri' => 'http://www.myconsumersite.com/oauth_callback',
        'application_uri' => 'http://www.myconsumersite.com/',
           'application_title' => 'John Doe\'s consumer site',
            'applicati-
on_descr' => 'Make nice graphs of all your data',
        'application_notes' => 'Bladibla',
        'application_type' => 'website',
        'application_commercial' => 0
        );

    $conn = new DB_connection();
    // Register the consumer
    $store = OAuthStore::instance('MySQL', array('conn' => $conn-
>connection));
    $key   = $store->updateConsumer($consumer, $user_id);

    // Get the complete consumer from the store
    $consumer = $store->getConsumer($key, $user_id);

    // Some interesting fields, the user will need the key and secret
    $consumer_id = $consumer['id'];
    $consumer_key = $consumer['consumer_key'];
    $consumer_secret = $consumer['consumer_secret'];

    echo "Consumer ID: " . $consumer_id . "\n";
    echo "Consumer key: " . $consumer_key . "\n";
    echo "Consumer secret: " . $consumer_secret . "\n";
}
?>
```